Coding : Jurnal Komputer dan Aplikasi
Volume 09, No. 02 (2021), hal 198-210 ISSN : 2338-493X

IMPLEMENTASI METODE *FAILOVER* SEBAGAI *BACKUP SERVER* PADA ARSITEKTUR *LOAD BALANCER*

Yolanda Bru Ginting[1], Uray Ristian[2]
[1,2]Jurusan Rekayasa Sistem Komputer, Fakultas MIPA Universitas Tanjungpura
Jalan Prof. Dr. H. Hadari Nawawi Pontianak
Telp./Fax : (0561) 577963
e-mail: [1]yolandabruginting@student.untan.ac.id, [2]eristian@siskom.untan.ac.id

Abstrak

Web server merupakan penunjang utama bagi berjalannya sebuah *website*. Teknologi *load balancing* yang merupakan salah satu metode yang dapat meningkatkan ketersediaan dan kecepatan kinerja *web server* juga dapat memiliki masalah dalam penerapannya. *Failover* merupakan kemampuan suatu sistem untuk berpindah ke *server* cadangan jika sistem utama mengalami kegagalan. Dalam penelitian ini *load balancing* dengan metode *failover* akan diimplementasikan pada sistem operasi Ubuntu. *Software* inti yang digunakan dalam penelitian ini adalah Nginx yang berfungsi sebagai *load balancer* sekaligus mendukung *failover*. Pengujian dilakukan dengan menggunakan perangkat lunak GoAccess untuk melihat data trafik pada sistem. Berdasarkan pengujian yang dilakukan, sistem yang dirancang berhasil membagikan beban permintaan dan dapat terus bekerja walaupun terjadi kegagalan pada *load balancer*. Rata-rata waktu respon di *web server* 1 selama 1358.2 ms, sedangkan di *web server* 2 selama 1355 ms. Berdasarkan rata-rata tersebut dapat disimpulkan bahwa waktu respon *web server* 2 lebih cepat dari pada *web server* 1. Pada pengujian *failover* saat percobaan ke-1 *master server* dalam keadaan mati yang berhasil masuk sebanyak 86 *request* dan gagal sebanyak 114 *request* dengan rata-rata waktu respon sebesar 1414 ms. Saat percobaan ke-2 kondisi *master server* aktif kembali jumlah *request* yang berhasil masuk sebanyak 200 *request* dengan rata-rata waktu respon selama 1306.5 ms. Hasil pengujian ini membuktikan bahwa sistem *load balancing* yang terpasang mampu melakukan *failover* untuk mengantisipasi kegagalan pada *server load balancer*.

Kata kunci— *Load Balancing, Failover,* Nginx, *Response Time, Backup Server*

1. PENDAHULUAN

Web server merupakan penunjang utama bagi berjalannya sebuah *website*. Semakin banyak *request* terhadap informasi melalui suatu situs *web* dapat membuat beban kerja yang lebih pada *web server* dan menjadi kurang optimal dalam ketersediaan informasi yang diminta. *Request* yang mencapai ribuan bahkan jutaan pada sebuah *single server* dapat menyebabkan kegagalan pada *server*. Solusi lain yang dapat diterapkan adalah teknologi *load balancing* yang berfungsi untuk mendistribusikan beban kerja saat *server* utama mengalami *overload request*. *Load balancing* merupakan aspek penting yang dapat mendistribusikan beban kerja ke banyak *server* secara optimal yang menghasilkan waktu tanggap yang baik, meningkatkan efisiensi penggunaan sumber daya, dan berdampak pada peningkatan performa secara keseluruhan. Dapat dikatakan bahwa penggunaan *load balancing* dapat mengatasi kegagalan yang terjadi pada sisi *web server*. Akan tetapi, jika *load balancing* mengalami kegagalan, seluruh layanan dapat terhenti. Oleh karena itu, diperlukan mekanisme *failover* yang dapat menjalankan fungsi *backup server* pada *load balancing* yang terpasang.

Terdapat beberapa penelitian yang telah dilakukan sebelumnya terkait implementasi *load balancing* maupun *failover*. Penelitian sebelumnya yang berjudul "Implementasi *Health Check Monitoring* pada *Load balancer* di Sisi *Web Server* Berbasis *Android*" melakukan penelitian terhadap arsitektur *load*

balancer yang di lengkapi sistem *monitoring* kesehatan dan kondisi *web server*. Berdasarkan hasil penelitian dengan adanya *load balancer request* yang masuk dapat di distribusikan dengan baik terhadap tiga buah *web server* dan dengan adanya *health check monitoring* kondisi *web server* dengan mudah di ketahui dan di tangani ketika mengalami masalah. Namun pada penelitian tersebut terdapat kelemahan pada sisi *load balancer* dimana hanya terdapat satu *server load balancer* yang menangani *request*, yang mana apabila terjadi kegagalan pada *server load balancer* tersebut maka *request* yang masuk tidak dapat diproses hingga ke *web server* [1].

Penelitian terkait selanjutnya membahas tentang metode *failover* dengan judul penelitian "Implementasi *High Availability Server* Menggunakan Metode *Load Balancing* dan *Failover* pada *Virtual Web Server Cluster*". *Server load balancer* yang digunakan sebanyak dua buah karena untuk menerapkan metode *failover* tersebut. Mode *failover* yang digunakan adalah mode aktif/pasif. Penggunaan mode aktif/pasif dapat berjalan dengan baik, akan tetapi hanya pada satu *load balancer* saja yang memiliki IP *floating* yaitu pada *load balancing master*. Didapatkan nilai rata-rata *downtime* yang diperoleh dari dua puluh lima hasil pengamatan yaitu sebesar 1992.8 ms, yang artinya memiliki *downtime* yang cukup lama sekalipun telah ditambahkan satu buah *server load balancer* [2].

Kemudian pada penelitian berikutnya dengan judul "Perancangan dan Pengujian *Failover* Menggunakan Nginx" dikatakan bahwa sistem yang dirancang dapat meningkatkan ketersediaan karena mampu melakukan *failover* saat terjadi kegagalan baik di sisi *load balancer* atau pun si sisi *backend server*. Namun pada penelitian ini penggunaan *load balancing* membuat sisi *database* menjadi lebih berat karena hanya menggunakan satu buah *database* sehingga saat menjalankan *request* yang memerlukan *query* yang berat, banyak *request* yang gagal [3].

Berdasarkan permasalahan pada penelitian sebelumnya maka akan dilakukan penelitian dengan judul "Implementasi Metode *Failover* Sebagai *Backup Server* pada Arsitektur *Load balancer*" sebagai solusi dalam ketersediaan *server* dan *load balancer* dapat menjalankan fungsinya dalam membagi beban *request*. Sehingga apabila *server* utama *load balancer* mengalami masalah maka *server backup* dapat menjalankan fungsi *load balancer* dalam mendistribusikan *request* ke *web server*.

Penelitian ini bertujuan untuk menerapkan mekanisme *failover* pada arsitektur *load balancer* dengan menggunakan Nginx dan menganalisa pengaruh mekanisme *failover* terhadap pembagian beban *request* dan waktu respon pada *web server*.

2. LANDASAN TEORI

2.1 *Load Balancing*

Load balancing adalah teknik untuk mendistribusikan beban trafik pada dua atau lebih jalur koneksi secara seimbang, agar trafik dapat berjalan optimal, memaksimalkan *throughput*, memperkecil waktu tanggap dan menghindari *overload* pada salah satu jalur koneksi. *Load balancing* digunakan pada saat sebuah *server* telah memiliki jumlah *user* yang telah memiliki maksimal kapasitasnya [3].

2.2 *Round Robin*

Algoritma *round robin* merupakan algoritma yang paling sederhana dan banyak digunakan oleh perangkat *load balancing*. Algoritma ini pada Nginx membagi beban secara bergiliran dan berurutan dari satu *server* ke *server* lain sehingga membentuk putaran namun, penggunaan *quantum time* sebagai pembatas waktu proses diabaikan dalam implementasi *load balancer* pada *web server* Nginx sehingga seluruh *request client* yang datang menuju *load balancer server* akan di bagi secara bergantian ke tiap *backend server* tanpa memperhatikan waktu proses sebelumnya dan proses dalam *server* [4].

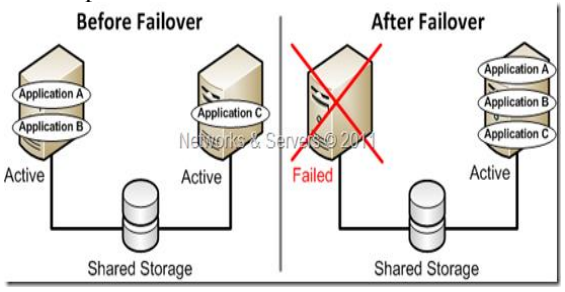
Tahapan pada algoritma *Round Robin* adalah dengan mendistribusikan *request client* ke *backend server* secara berurutan. Pada penjadwalan tipe *round-robin*, *load balancer* mendistribusikan *client request* sama rata ke seluruh *real server* tanpa memperdulikan kapasitas *server* ataupun beban *request*. Jika ada dua *real server* (A, B), maka *request* 1 akan diberikan *load balancer* kepada *server* A, *request* 2 ke *server* B, dan *request* 3 kembali ke *server* A. Mekanisme ini dapat dilakukan jika seluruh *real server* menggunakan spesifikasi komputer yang sama. Konsep dasar dari

algoritma ini adalah dengan menggunakan *time-sharing*. Pada dasarnya algoritma ini sama dengan FCFS (*First Come First Service*), hanya saja bersifat *preemptive* yaitu proses akan dihentikan, dipindahkan ke *node* lain dan proses akan berlanjut. Setiap proses mendapatkan waktu CPU yang disebut dengan waktu *quantum* (*quantum time*) untuk membatasi waktu proses. Setelah waktu habis, proses ditunda dan ditambahkan pada *ready queue*. Algoritma *round robin* digunakan pada *server* untuk menyeimbangkan beban pada beberapa *server* [4].

2.3 *Failover*

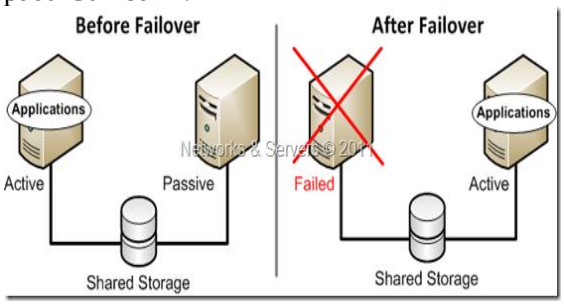
Failover yang juga disebut *high-availability cluster* pada umumnya diimplementasikan untuk tujuan meningkatkan ketersediaan layanan yang disediakan oleh klaster. Elemen klaster memiliki *node-node* redundan yang akan digunakan untuk menyediakan layanan ketika salah satu komponen mengalami kegagalan. Dibutuhkan dua buah *node* sebagai syarat minimun suatu klaster untuk dapat melakukan redudansi [2]. *Failover* memiliki dua macam mode yaitu mode *active/active* dan mode *active/passive*.

Pada mode *active/active*, semua *node* berstatus aktif dengan menjalankan masing-masing aplikasi atau proses yang merupakan beban kerja masing-masing *node*. Apabila satu *node* aktif mengalami kegagalan, *node* aktif lain akan mengambil alih beban kerja *node* yang gagal tersebut sehingga *node* aktif yang masih berfungsi menjalankan seluruh aplikasi dan proses yang ada [2]. Tujuan dari *active/active failover* ini adalah untuk mencapai *load balancing*. Yang membedakan mode ini dengan *load balancing cluster* yaitu adanya konfigurasi untuk *redundancy* diantara kedua *node* yang aktif. Arsitektur *active/active failover* dapat dilihat pada Gambar 1.



Gambar 1. *Active/Active Failover*

Sedangkan pada mode *active/passive node* yang satu menjadi komponen atau *node* aktif dan yang lainnya pasif. *Node* aktif bertugas untuk melakukan eksekusi terhadap aplikasi atau tugas tertentu, sedangkan *node* pasif berstatus *stand by* dengan tidak melakukan tugas apapun sampai mendeteksi bahwa terdapat masalah pada *node* utama/aktif. Pada saat *node* utama mengalami kegagalan, *node* pasif akan mengambil alih tugas yang tadinya dilakukan oleh *node* utama [2]. Arsitektur *active/passive failover* ditunjukkan pada Gambar 2.



Gambar 2. *Active/Passive Failover*

Metode *failover* yang digunakan pada penelitian ini menggunakan mode *active*/*passive* dimana salah satu *node* (*server*) berstatus aktif dengan menjalankan semua aplikasi atau proses yang merupakan beban kerjanya. Apabila *node* yang aktif mengalami kegagalan, maka *node passive* akan mengambil alih beban kerja *node* yang gagal tersebut sehingga *node passive* yang *standby* dapat menjalankan seluruh aplikasi dan proses yang ada.

2.4 *Virtual Private Server*

Virtual Private Server (VPS) adalah teknologi *server side* tentang sistem operasi dan perangkat lunak yang memungkinkan sebuah mesin dengan kapasitas besar dibagi ke beberapa *virtual* mesin. Tiap *virtual* mesin ini melayani sistem operasi dan perangkat lunak secara mandiri dan dengan konfigurasi yang cepat. VPS juga dapat diartikan sebagai sebuah metode untuk mempartisi atau membagi sumber daya atau *resource* sebuah *server* menjadi beberapa *server virtual*. *Server virtual* tersebut memiliki kemampuan menjalankan *operating system* sendiri seperti layaknya sebuah *server*. Bahkan pengguna dapat me-*reboot* sebuah *server virtual* secara terpisah (tidak harus me-*reboot server* utama). VPS

dapat dikendalikan dengan *remote access desktop* atau biasa disebut pengendali jarak jauh, dengan menggunakan aplikasi seperti Putty untuk yang menggunakan OS Windows dan Terminal untuk Linux [5].

2.5 *Web Server*

Web server merupakan perangkat lunak yang melayani permintaan HTTP dari *web browser* dan mengirimkan kode-kode dinamis ke *server* aplikasi. Fungsi utama *server* atau *web server* adalah untuk melakukan atau akan mentransfer berkas permintaan pengguna melalui protokol komunikasi yang telah ditentukan sedemikian rupa. Pemanfaatan *web server* berfungsi untuk mentransfer seluruh aspek pemberkasan dalam sebuah halaman *web* termasuk yang di dalam berupa teks, video, gambar dan banyak lagi [6].

2.6 Nginx *Web Server*

Nginx adalah salah satu dari sebagian perangkat lunak untuk *server* yang diciptakan untuk mengatasi masalah C10K. Dalam masalah ini, C10K adalah ketika *web server* diminta untuk menangani sepuluh ribu *client* secara bersamaan. Tidak seperti perangkat lunak *server* lainnya, Nginx tidak bergantung kepada *thread* untuk melayani *client*. Penggunaan Nginx pada penelitian ini dikarenakan konsep *asynchronous* nya yang terkenal lebih cepat dalam pelayanan *web server* dan *real-time*. Konsep *asynchronous* merupakan proses yang terdapat pada Nginx yang digunakan untuk mengeksekusi kode perintah yang diberikan sebelum perintah tersebut selesai di jalankan [7].

2.7 GoAccess

GoAccess adalah program penganalisis *web server log* yang interaktif dan *real-time* yang dapat menganalisis dan melihat *web server log* dengan cepat. GoAccess bersifat *open-source* dan berjalan sebagai baris perintah di sistem operasi Unix/Linux. GoAccess dapat memberikan laporan statistik HTTP (*web server*) singkat dan bermanfaat untuk administrator Linux dengan cepat. GoAccess juga dapat menangani format *web server log* dari Apache dan Nginx.

GoAccess mem-*parsing* dan menganalisis format *web server log* yang diberikan dalam opsi yang lebih disukai termasuk CLF (*Common Log Format*), format W3C (IIS), dan *host virtual* Apache, lalu menghasilkan keluaran data ke terminal. Meskipun keluaran terminal adalah keluaran *default*, GoAccess memiliki kemampuan untuk menghasilkan laporan dalam format *real-time* HTML yang lengkap dan mandiri (cocok untuk analitik, pemantauan, dan visualisai data), serta laporan dalam format JSON dan CSV (*Comma Separated Value*) [8].

3. METODE PENELITIAN

Untuk kelancaran penelitian, dibuat sebuah diagram alir penelitian yang dapat membantu peneliti dalam menentukan langkah-langkah penelitian yang akan dilakukan. Penelitian dimulai dengan melakukan studi literatur yaitu mengumpulkan referensi pendukung penelitian, dilanjutkan dengan menentukan metode pengumpulan data yang digunakan. Setelah langkah tersebut dilakukan, penelitian dilanjutkan dengan menganalisis kebutuhan penelitian, selanjutnya dilakukan perancangan sistem dan implementasi hasil perancangan. Setelah itu dilanjutkan dengan pengujian. Apabila pengujian telah berhasil selanjutnya analisis hasil pengujian dan pembahasan. Diagram alir penelitian dapat dilihat pada Gambar 3.

Gambar 3. Diagram Alir Metode Penelitian

3.1 Studi Literatur

Studi literatur yang dilakukan yaitu dengan mencari referensi teori yang relevan dengan permasalahan yang ada untuk membangun sistem. Referensi yang dicari berkaitan dengan pembangunan sistem *load balancer* pada *web server* dengan menambahkan metode *failover* pada *server load balancer*. Literatur yang digunakan berupa jurnal, *e-book*, artikel dan informasi yang didapat melalui pertanyaan-pertanyaan.

3.2 Metode Pengumpulan Data

Metode pengumpulan data yang dilakukan adalah observasi. Metode observasi adalah metode pengamatan secara langsung kegiatan yang sedang dilakukan pada penelitian ini. Obyek pada penelitian ini adalah arsitektur *load balancer*. Observasi yang dilakukan terhadap obyek penelitian tersebut adalah menemukan data yang dapat digunakan untuk proses implementasi sistem dan pengujian. Data yang dikumpulkan adalah parameter keberhasilan yang digunakan pada penelitian terdahulu yang dapat dijadikan contoh untuk melakukan pengujian seperti jumlah *request* dan *response time*.

3.3 Analisa Kebutuhan

Analisa kebutuhan dilakukan untuk menentukan kebutuhan-kebutuhan perangkat keras (*hardware*) dan perangkat lunak (*software*) sebagai bahan untuk perancangan dan pembuatan sistem dari permasalahan yang ada sebagai penyelesaian.

3.3.1 Kebutuhan Perangkat Keras

Perangkat keras yang dibutuhkan yaitu satu buah laptop dengan spesifikasi sebagai berikut:

1. *Prosessor* : Minimal Intel ® Celeron ® N2820 (up to 2.39 GHz, 1 MB L2 *cache*)
2. *Storage* : 128 GB SSD
3. *Memory* : 2 GB

3.3.2 Kebutuhan Perangkat Lunak

Perangkat lunak yang dibutuhkan dalam penelitian ini adalah sebagai berikut:

1. *Virtual Private Server*
2. *Web server* Nginx pada *load balancer server*
3. *Web server* Apache2 pada *backend server*
4. Sistem Operasi Ubuntu *Server* 18.04 LTS pada *server*
5. Sistem Operasi Windows 10 64 bit pada *client*
6. *Web Browser* sebagai media untuk mengakses tampilan aplikasi selama pengerjaan

3.4 Perancangan Sistem

Pada bagian ini akan dijelaskan tentang tahap perancangan sistem yang dimulai dengan merancang arsitektur sistem secara keseluruhan, perancangan arsitektur *cluster load balancer server* dan mekanisme *failover*, penerapan metode *round robin* pada *load balancer* dan *failover*, perancangan sistem pemantauan, perancangan aplikasi *website monitoring*, perancangan antar muka aplikasi, dan perancangan pengujian.

3.5 Implementasi

Pada bagian ini akan dilakukan implementasi atau proses pengerjaan sistem berdasarkan rancangan yang telah dibuat. Implementasi yang akan dilakukan adalah implementasi sistem yang terdiri dari implementasi *cluster load balancer server*, implementasi mekanisme *failover*, dan implementasi aplikasi pemantauan. Kemudian dilanjutkan dengan implementasi aplikasi *website monitoring*, dan implementasi kode program aplikasi *website monitoring*.

3.6 Pengujian Sistem

Pada bagian ini akan dilakukan pengujian terhadap sistem yang telah dibangun apakah sesuai dengan rancangan awal atau tidak. Jika pengujian berhasil, maka akan dilanjutkan pada tahap analisa dan pembahasan. Jika pengujian tidak berhasil maka akan kembali ke tahap perancangan untuk mengecek kembali perancangan yang dibuat. Pada penelitian ini akan dilakukan tiga tahapan pengujian terhadap sistem yang telah dibangun dan disertai bentuk analisa yang akan dilakukan sebagai berikut:

1. Pengujian pembagian beban *request* pada *load balancer*.

Pengujian pembagian beban *request* dilakukan untuk menguji metode *load balancer* yang telah diterapkan dengan menggunakan algoritma *round robin* pada Nginx. Parameter yang digunakan pada pengujian ini adalah jumlah *request* yang terbagi ke masing-masing

web server. Analisa dilakukan pada sisi *server* dan aplikasi *website monitoring*.

2. Pengujian *failover* terhadap *load balancer* saat *master server* tidak aktif.

Pengujian *failover* dilakukan untuk menguji metode *failover* yang telah diterapkan dengan menggunakan Nginx. Parameter yang digunakan pada pengujian ini adalah jumlah *request* yang berhasil dan jumlah *request* yang gagal pada saat *master server* dimatikan dan koneksi *server* dapat beralih dari *master server* ke *backup server*. Analisa dilakukan pada sisi *server* dan aplikasi *website monitoring*.

3. Pengujian aplikasi *website monitoring* dalam menampilkan data hasil pengujian.

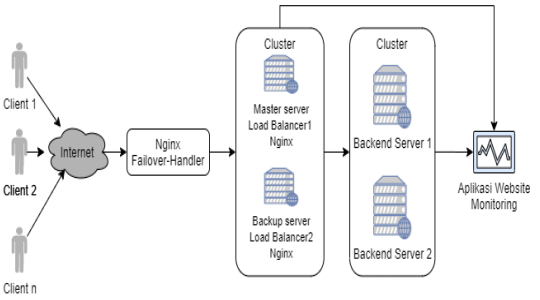
Pengujian aplikasi *website monitoring* dilakukan untuk menguji sistem *monitoring* yang telah dibuat dalam menampilkan data hasil pengujian. Parameter yang digunakan pada pengujian ini adalah data dari *server* dapat ditampilkan pada aplikasi, pembagian *request* merata pada masing-masing *server*. Analisa dilakukan pada sisi aplikasi.

4. HASIL DAN PEMBAHASAN

Pada penelitian ini akan dibangun sistem *backup server* dengan mengimplementasikan metode *failover* pada *load balancer server* yang dapat meningkatkan ketersediaan layanan *web server*. Mode *failover* yang digunakan pada penelitian adalah mode *active*/*passive* dimana salah satu *node* (*server*) berstatus aktif sedangkan *node* lainnya dalam keadaan *standby*. *Node* yang *standby* akan bekerja ketika *node* aktif mengalami kegagalan sistem. Metode *failover* akan dikonfigurasikan ke dalam dua buah *server load balancer* di mana pada arsitektur *load balancer* terdapat *multiple server* untuk membagi beban kerja *web server*.

4.1 Arsitektur Sistem

Perancangan arsitektur sistem dimulai dengan merancang arsitektur sistem secara keseluruhan dengan memegang konsep utama *load balancer*, dimana *load balancer* sebagai penghubung antara jaringan dan *server* yang melayani *request*. Dalam arsitektur ini dibutuhkan lima buah *server*. Sistem *load balancer* dilakukan oleh dua buah *server* yang didefinisikan dengan nama *Master Server* sebagai *load balancer* 1 dan *Backup Server* sebagai *load balancer* 2, mekanisme *failover* dilakukan oleh sebuah *server* yang didefinisikan dengan nama *Failover Handler*, dan aplikasi *server* dilakukan oleh dua buah *server* yang didefinisikan dengan nama *backend server* 1 dan *backend server 2*. Rancangan arsitektur sistem secara keseluruhan dapat dilihat pada Gambar 4.



Gambar 4. Arsitektur Keseluruhan Sistem

Pada penelitian ini arsitektur sistem dibangun dalam lingkungan *virtual* yang didapatkan dari penyedia layanan *virtual private server*. *Provider virtual private server* yang digunakan adalah Upcloud untuk *failover handler server*, CloudKilat untuk *load balancer server* dan *Amazon Web Service* (AWS) untuk *web server*. Adapun spesifikasi *server* yang digunakan ditunjukkan pada Tabel 1.

Tabel 1. Spesifikasi *Virtual Private Server*

No	Jenis *Server*	CPU	RAM	Storage	Provider
1	*Failover Handler*	1vCPU	1 GB	25 GB SSD	Upcloud
2	*Load balancer* 1	1vCPU	1 GB	20 GB	Cloud Kilat

Tabel 1. Spesifikasi *Virtual Private Server* (lanjutan)

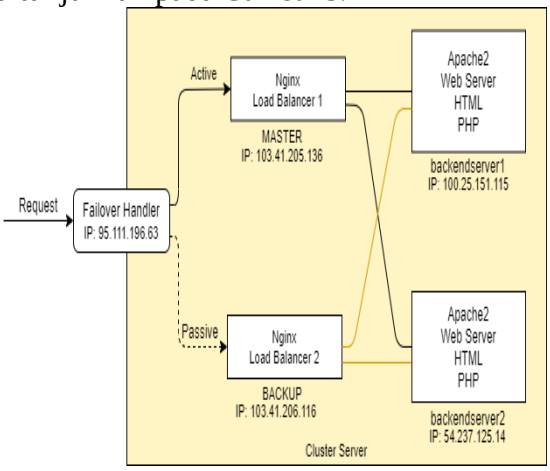
No	Jenis *Server*	CPU	RAM	Storage	Provider
3	*Load balancer* 2	1vCPU	1 GB	20 GB	Cloud Kilat
4	*Web Server* 1	1vCPU	1 GB	8 GB	AWS
5	*Web Server* 2	1vCPU	1 GB	8 GB	AWS

Setiap *server* disertai dengan alamat IP yang digunakan oleh *virtual private server*. Alamat IP ini terdiri dari alamat IP *server failover handler*, *load balancer* 1, *load balancer* 2, *web server* 1, dan *web server* 2. Alamat IP tersebut ditampilkan pada Tabel 2.

Tabel 2. Nama *Server* dan Alamat IP

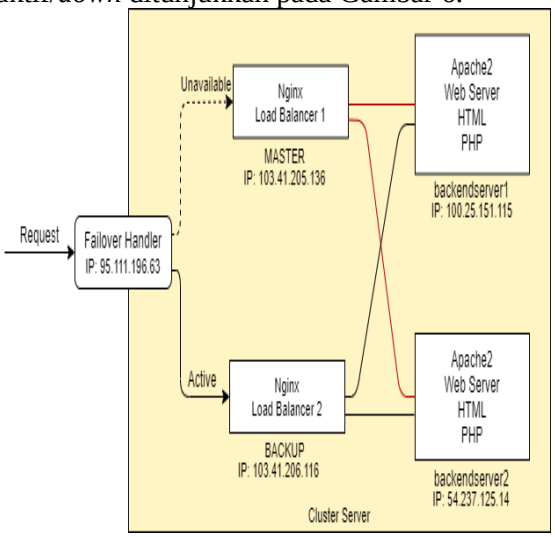
Nama *Server*	**Alamat IP**
Failover Handler	95.111.196.63
Master server: *Load balancer* 1	103.41.205.136
Backup server: *Load balancer* 2	103.41.206.116
Backend server 1: *Web Server* 1	100.25.151.115
Backend server 2: *Web Server* 2	54.237.125.14

Setelah perancangan arsitektur selesai dilakukan, selanjutnya dilakukan penerapkan mekanisme *failover* pada *server Failover Handler*. Dengan berpegang pada konsep *failover* sebagai mekanisme pindah jalur, maka terdapat dua gambaran kondisi *load balancer server* saat diberikan *request*, yaitu saat *load balancer* 1 dalam kondisi aktif dan saat *load balancer* 1 tidak aktif. Pada saat kondisi *master server* sedang aktif *request* yang masuk dari klien akan di terima oleh IP *failover handler* yang mana didalam *server* tersebut telah diberikan konfigurasi menggunakan *ngx_http_upstream_module* yang merupakan modul bawaan dari Nginx untuk mengecek koneksi antara *master server* dan *backup server*. Cara kerja mekanisme *failover* pada saat kondisi *master server* sedang aktif ditunjukkan pada Gambar 5.



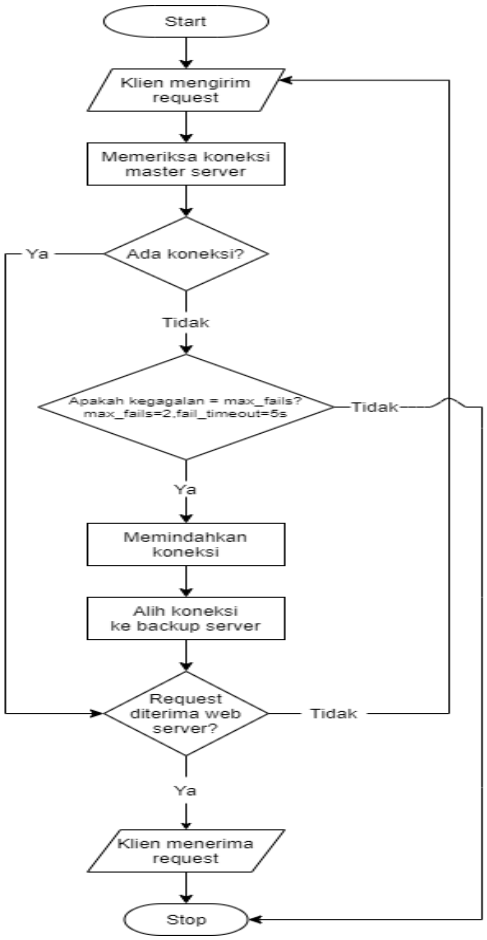
Gambar 5. Kondisi *Master Server* Aktif

Setelah *request* diterima maka *failover handler* akan mengecek kondisi *cluster load balancer server* menggunakan metode pengecekan *request time-out* selama 5 detik. Jika *failover handler* mendeteksi bahwa *master server* sedang aktif, maka *request* akan dikirim ke *master server* yang kemudian membagi *request* ke *web server*. Namun, apabila *failover handler* mendeteksi dua kali *master server* terbaca *request time-out*, maka *request* akan dialihkan ke *backup server* yang dalam kondisi *standby* sehingga *request* dapat tetap dilayani oleh *load balancer* dan dikirim ke *web server*. Kondisi tersebut merupakan kondisi pada saat *master server* tidaf aktif/*down* dan *failover* melakukan fungsinya untuk mem-*backup server*. *Master server* dalam kondisi tidak aktif/*down* ditunjukkan pada Gambar 6.



Gambar 6. Kondisi *Master Server* Tidak Aktif

Proses dan cara kerja sistem yang akan dibangun digambarkan dalam bentuk diagram alir sistem. Diagram alir sistem menjelaskan cara kerja sistem yang dimulai dengan klien mengirim *request* ke *server*, kemudian *failover handler* memeriksa koneksi *master server* berdasarkan algoritma *round robin* pada Nginx. Apabila *master server* terdeteksi aktif, maka *request* diterima oleh *web server* kemudian mengirimkan kembali *request* tersebut kepada klien dan sistem selesai. Apabila *master server* terdeteksi tidak aktif, maka sistem akan memeriksa apakah kegagalan sama dengan maksimal *time-out* yang telah ditentukan, apabila sama dengan maksimal *time-out* yang telah ditentukan, maka sistem akan otomatis memindahkan koneksi dari *master server* ke *backup server* dan meneruskan *request* ke *web server*, jika tidak sama dengan maksimal *timeout* yang telah ditentukan, maka semua proses dihentikan. Diagram alir sistem ditunjukkan pada Gambar 7.



Gambar 7. Diagram Alir Sistem

4.2 Implementasi *Cluster Load balancer Server*

Implementasi *cluster load balancer server* dilakukan pada kedua *server load balancer* dengan isi konfigurasi yang sama pada *file /etc/*nginx*/sites-enabled/"namafilemasing-masingserver"*. Implementasi ini dikonfigurasi dengan menggunakan perangkat lunak Xshell7. Implementasi ini dilakukan untuk menerima *request* yang dikirimkan pada *server load balancer*. Implementasi konfigurasi *server load balancer* 1 dapat dilihat berdasarkan konfigurasi *file etc/*nginx*/sites-enabled/app.serverku.local* pada Gambar 8.

```
 GNU nano 2.9.3 /etc/nginx/sites-enabled/app.serverku.local

upstream backend {
        server 100.25.151.115:80;
        server 54.237.125.14:80;
        }

server {
        listen 80;
        server_name     yolanda.greatlikemasterstronglike.monster;
        access_log      /var/log/nginx/app_access.log;
        error_log       /var/log/nginx/app_error.log;
        location / {
                proxy_pass http://backend;
                include proxy_params;
        }
}
```

Gambar 8. Konfigurasi *Master Server/Load Balancer* 1

Pada Gambar 8 diperlihatkan konfigurasi penggunaan *load balancer* pada *server* "103.41.205.136" menggunakan perangkat lunak Xshell7. *Block server* menunjukkan setiap *server* yang datang ke alamat "103.41.205.136:80/" akan dilanjutkan ke alamat yang telah didefinisikan pada kelompok *upstream backend*. *Upstream backend* berisi 2 alamat IP dari anggota *backend server* yaitu '100.25.151.115:80' dan '54.237.125.14:80'. Dua alamat tersebut merupakan *cluster server* yang saling berbagi beban dalam menyelesaikan *request* yang datang pada *server load balancer*. Konfigurasi yang sama juga dilakukan menggunakan perangkat lunak Xshell7 pada *backup server* yang dapat dilihat pada Gambar 9.

```
        /etc/nginx/sites-enabled/app.serverbackup.local

upstream backend {
        server 100.25.151.115:80;
        server 54.237.125.14:80;
}

server {
        listen 80;
        server_name     yolanda.greatlikemasterstronglike.monster;
        access_log      /var/log/nginx/app_access.log;
        error_log       /var/log/nginx/app_error.log;
        location / {
                proxy_pass http://backend;
                include proxy_params;
        }
}
```

Gambar 9. Konfigurasi Backup *Server/Load Balancer* 2

4.3 Implementasi *Failover Handler*

Pada penelitian ini metode *failover* diimplementasikan dengan menggunakan *ngx_http_upstream_module* yang merupakan modul bawaan dari Nginx. Modul ini akan mendefinisikan kelompok *server* yang terdiri dari dua alamat *server* yaitu *load balancer* 1 dan *load balancer* 2. Pada alamat *server load balancer* 1 diberikan konfigurasi *health check max_fails*=2 *fails_timeout*=5s sedangkan pada alamat *server load balancer* 2 diberikan konfigurasi fungsi *backup*. Konfigurasi dilakukan menggunakan perangkat lunak Xshell7. Implementasi *failover handler* dapat dilihat pada Gambar 10.

```
        /etc/nginx/sites-enabled/failover-handler.local
upstream backend {
        server 103.41.205.136:80 max_fails=2 fail_timeout=5s;
        server 103.41.206.116:80 backup;
        }

server {
        listen 80;
        server_name     yolanda.greatlikemasterstronglike.monster;
        access_log      /var/log/nginx/app_access.log;
        error_log       /var/log/nginx/app_error.log;
        location / {
                proxy_pass http://backend;
                include proxy_params;
        }
}
```

Gambar 10. Konfigurasi *Failover Handler*

4.4 Implementasi Sistem Pemantauan

Implementasi sistem pemantauan dilakukan berdasarkan analisis kebutuhan dan perancangan pada poin 3.3 dan 3.4. Pada tahap ini akan dijelaskan tentang proses implementasi *script* perintah GoAccess dan Cronjob pada masing-masing *server* yang akan di-*monitoring*. *Server* yang akan di-*monitoring* adalah *web server* 1 dan *web server* 2. Pengumpulan data yang dilakukan sistem yaitu pembacaan *log* pada *web server* yang sebelumnya telah dikonversi kedalam bentuk *file* JSON. Tugas dari sistem pemantauan adalah melakukan *static routing* agar *file* dalam format JSON tersebut dapat diakses oleh administrator melalui *request* HTTP berdasarkan URL. Pada Gambar 11 dapat dilihat *request* HTTP ke URL *http://100.25.151.115/report.json* berhasil mendapatkan data *log web server* Apache2 dalam bentuk JSON API yang nanti akan digunakan atau dipanggil oleh fungsi tertentu.

Gambar 11. Data Hasil Konversi *Log Web Server* Ke dalam Format JSON

Script yang dijalankan untuk mendapatkan data *log* tersebut merupakan perintah dari GoAccess yang berfungsi untuk menampilkan data secara *real-time*. Pertama-tama membuat *file* dalam format HTML dengan perintah *sudo goaccess access.log –o /var/www/backendserver1/report.html –log-format=COMBINED*, kemudian men-*generate* HTML kedalam format JSON dengan perintah *sudo goaccess access.log –o /var/www/backendserver1/report.json –log-format=COMBINED*. *Log* tersebut tidak secara otomatis dibuat dalam format waktu maka diperlukan sebuah operasi yang dapat dijalankan permenit untuk memecahkan masalah tersebut. Operasi tersebut dilakukan menggunakan Cronjob yang dikonfigurasikan pada *web server* 1 dan *web server* 2 menggunakan *nano text editor* untuk mengatur riwayat *access log* pada *server* dalam rentang waktu 1 menit dengan perintah ***** *goaccess /var/log/apache2/access,log –o/var/www/backendserver1/report.json –log-format=COMBINED*.

Log tersebut mengumpulkan berbagai data hasil *request* ke *web server* seperti tanggal dan waktu akses, total *request*, *request* yang berhasil dan yang gagal, data pengunjung, dan lain sebagainya yang dapat membantu analisis hasil pengujian. Pada penelitian ini data yang dianalisis berdasarkan *log* tersebut adalah total jumlah *request* yang masuk, *request* yang berhasil, *request* yang gagal, dan waktu respon.

4.5 Pengujian *Load Balancer*

Pengujian *load balancer* dilakukan untuk mendapatkan data pembagian beban *request* dari *load balancer* terhadap *web server*, yang mana data tersebut akan ditampilkan pada aplikasi *website* sebagai hasil pengujian. Pengujian dilakukan dengan menguji 5 sampel *request*. Pengujian ini dilakukan dengan mengirimkan *request* ke *server load balancer* kemudian hasil *request* yang telah dikirimkan diproses dengan menggunakan *tool* GoAccess. Tampilan hasil pengujian hanya akan diwakilkan 1 sampel pengujian dari 5 sampel *request* dan sisanya ditampikan dalam bentuk tabel.

Sebelum melakukan pengujian terlebih dahulu dilakukan proses penghapusan *log* pada *web server* 1 dengan menggunakan perintah *sudo echo > /var/log/apache2/access.log* dan *web server* 2 dengan perintah yang sama untuk memastikan data yang masuk sesuai dengan data sampel *request* yang dimasukan. Kemudian sampel 1 sebanyak 100 *request* dikirimkan pada *server failover handler* melalui *server client*. Proses ini dilakukan menggunakan *tool* Apache Benchmark dengan perintah *ab –n 100 –c http://95.111.196.63/*. Selanjutnya dilakukan pengujian dengan menjalankan perintah GoAccess pada *web server* 1 dengan perintah *sudo goaccess access.log –o/var/www/backendserver1/report.html –log-format=COMBINED* dan pada *web server* 2 dengan perintah *sudo goaccess access.log –o/var/www/backendserver2/report.html –log-format=COMBINED*. Hasil pengujian sampel

request 1 yang telah dikirim kemudian ditampilkan pada aplikasi *website monitoring*. Tampilan berupa tabel dan grafik jumlah *request* serta *response time*. Hasil pengujian *load balancer* ditampilkan pada Gambar 13.


TABEL HASIL PENGUJIAN

No	Nama Server	Request Masuk	Response Time (ms)
1	Server 1	50	1450
2	Server 2	50	1221

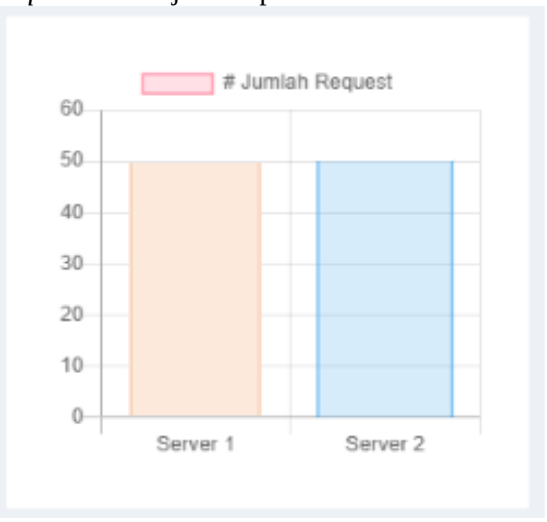
Gambar 13. Tampilan Hasil Pengujian Sampel

Berdasarkan pengujian dari sampel *request* 1 dengan memberikan 100 *request* ke *server* dapat diterima dengan baik oleh *web server* 1 sebanyak 50 *request* dengan *response time* 1450 ms dan *web server* 2 sebanyak 50 *request* dengan *response time* 1221 ms. Berdasarkan hasil pengujian tersebut *request* yang diterima oleh *web server* dapat terbagi dengan rata, namun terdapat perbedaan waktu respon dari masing-masing *server*. Waktu respon pada *web server* 1 lebih lama dari pada waktu respon *web server* 2. Hasil pengujian *sample request* 1 juga ditampilkan dalam bentuk grafik seperti pada Gambar 14 dan 15. Selanjutnya hasil pengujian pada sampel *request* 2 sampai sampel *request* 5 di jelaskan pada Tabel 3.

Tabel 3. Hasil Pengujian *Load balancer*

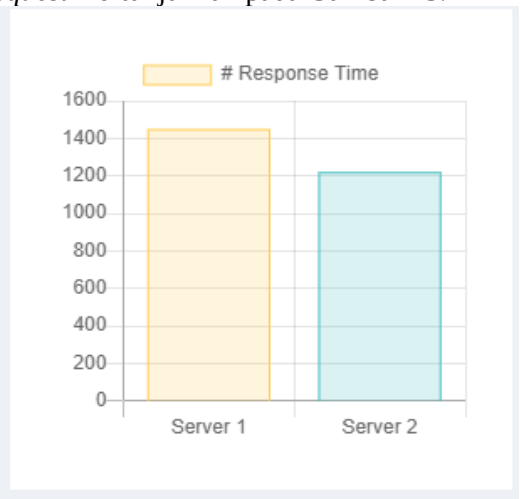
Nama Sampel	**Jumlah *Request Web Server* 1**	**Jumlah *Request Web Server* 2**	***Response Time Web Server* 1 (ms)**	***Response Time Web Server* 2 (ms)**
Sampel *request* 1	50	50	1450	1221
Sampel *request* 2	100	100	1320	1219
Sampel *request* 3	150	150	1254	1832
Sampel *request* 4	200	200	1404	1206
Sampel *request* 5	250	250	1363	1297

Grafik jumlah *request* pada pengujian sampel *request* 1 ditunjukkan pada Gambar 14.



Gambar 14. Grafik Jumlah *Request* Sampel *Request* 1

Grafik *response time* pengujian sampel *request* 1 ditunjukkan pada Gambar 15.

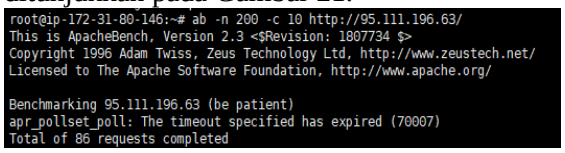


Gambar 15. Grafik *Response Time* Sampel *Request* 1

4.6 Pengujian Mekanisme *Failover* pada *Load Balancer*

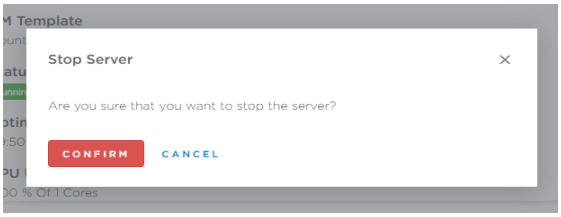
Pengujian dilakukan pertama kali dengan mengirimkan *request* menggunakan *command line* Apache Benchmark berikut: *ab –n 200 –c 10 http://95.111.196.63/*, sebanyak 200 *request* dikirimkan kepada IP *address 95.111.196*.63 yang akan diakses sebanyak 10 kali dalam waktu yang bersamaan. Kemudian pada pertengahan uji *master server* yang aktif sengaja dimatikan dan memaksa terjadinya alih koneksi ke *backup server*. Pemberian beban *request* dan sekaligus memperlihatkan jumlah

request yang berhasil diterima oleh *web server* ditunjukkan pada Gambar 21.



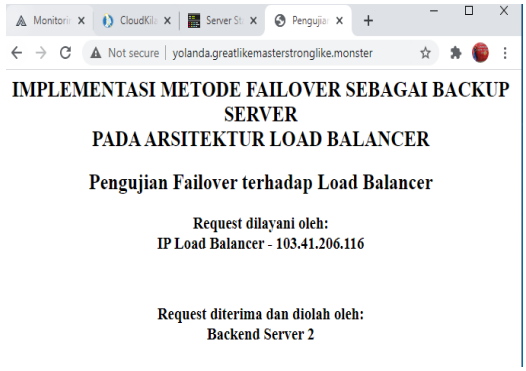
Gambar 21. Pemberian Beban *Request*

Kemudian pada halaman *client* CloudKilat, *master server* dimatikan dengan langkah seperti yang ditunjukkan pada Gambar 22.



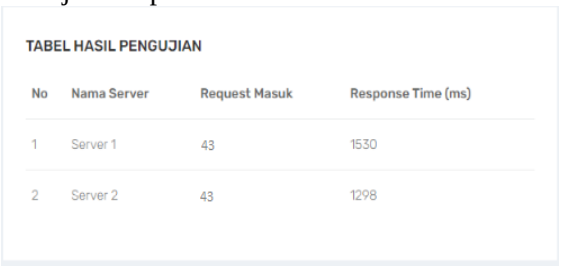
Gambar 22. Proses Mematikan *Master Server Load balancer*

Kemudian pada halaman pengujian yang dibuat dengan HTML murni dilihat perpindahan alamat IP *server* yang melayani *request* seperti yang ditunjukkan pada Gambar 23.



Gambar 23. Pengujian Perpindahan IP *Load Balancer*

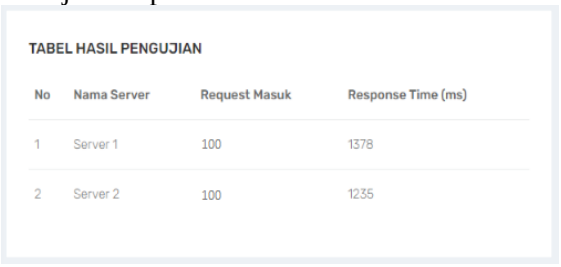
Kemudian untuk melihat jumlah *request* yang masuk pada *web server* dilakukan analisa pada aplikasi *website monitoring*. Pada pengujian percobaan ke-1 pada saat *master server* dalam kondisi *down request* yang berhasil masuk hanya sebanyak 86 *request* yang masih dapat ditangani oleh *load balancer* 1. *Load balancer* masih dapat bekerja dengan membagi beban *request* ke *web server* 1 sebanyak 43 *request* dengan waktu respon selama 1530 ms dan pada *web server* 2 sebanyak 43 *request* dengan waktu respon selama 1298 ms. Hasil percobaan ke-1 ditunjukkan pada Gambar 24.


TABEL HASIL PENGUJIAN

No	Nama Server	Request Masuk	Response Time (ms)
1	Server 1	43	1530
2	Server 2	43	1298

Gambar 24 Hasil Pengujian Percobaan Ke-1

Kemudian pada pengujian percobaan ke-2 pada saat *master server* kembali dalam kondisi aktif *request* yang masuk dapat diterima dengan lengkap dan *load balancer* dapat membagi rata *request* tersebut ke *web server* 1 sebanyak 100 *request* dengan waktu respon selama 1378 ms dan pada *web server* 2 sebanyak 100 *request* dengan waktu respon selama 1235 ms. Hasil percobaan ke-2 ditunjukkan pada Gambar 25.


TABEL HASIL PENGUJIAN

No	Nama Server	Request Masuk	Response Time (ms)
1	Server 1	100	1378
2	Server 2	100	1235

Gambar 25. Hasil Pengujian Percobaan Ke-2

Berdasarkan pengujian yang telah dilakukan dan hasil yang didapatkan kemudian dihitung rata-rata waktu respon untuk melihat perbandingan waktu respon dari setiap percobaan. Rata-rata waktu respon pada percobaan ke-1 selama 1414 ms dan rata-rata waktu respon pada percobaan ke-2 adalah selama 1306,5 ms. Hasil pengujian mekanisme *failover* dijelaskan pada Tabel 4.

Tabel 4. Percobaan Pengujian Mekanisme *Failover*

Percobaan ke-	Total Jumlah *Request*	*Request* Berhasil	*Request* Gagal	*Response Time (ms)*	Kondisi *Load Balancer*
1	200 *request*	86	114	1298	*Master server* tidak aktif/*down*
2	200 *request*	200	0	1235	*Master server* aktif

4.7 Pengujian Aplikasi *Website Monitoring*

Pada pengujian ini dilakukan pengujian terhadap aplikasi *website monitoring* untuk melihat keberhasilan sistem pemantauan mengambil data dari *server*. Parameter

keberhasilan pada pengujian ini adalah aplikasi dapat menampilkan data jumlah *request* yang terbagi ke *web server* dapat terbagi dengan rata. Hasil pengujian ditampilkan dalam bentuk gambar seperti pada Gambar 13 dan grafik seperti pada Gambar 14 dan 15. Pengujian dilakukan bersamaan dengan pengujian *load balancer*.

4.8 Pembahasan

Penelitian ini bertujuan untuk menerapkan mekanisme *failover* pada arsitektur *load balancer*. Metode *failover* yang digunakan merupakan mode *active*/*passive* dimana *master server* sebagai *server* utama *load balancer* dan *backup server* sebagai *server* kedua untuk melayani *request* yang datang dari *client* apabila *master server* mengalami kegagalan. Pada penelitian ini *failover* diimplementasikan menggunakan *ngx_http_upstream_module* yang merupakan modul bawaan dari Nginx. *Failover* didefinisikan pada satu alamat *server* yaitu *server failover handler*. Mekanisme *failover* yang diterapkan menggunakan Nginx dapat bekerja terhadap *load balancer* karena fungsi *load balancer* sekaligus mendukung terjadinya *failover*.

Berdasarkan pengujian yang telah dilakukan terdapat beberapa poin yang menjadi pembahasan, diantaranya:

1. Pada pengujian *load balancer* dengan 5 sampel *request* yang dikirimkan *request* dapat terbagi rata pada masing-masing *server*. Analisa hasil pengujian dilakukan pada aplikasi *website monitoring*. Rata-rata waktu respon pada *web server* 1 selama 1358.2 ms sedangkan pada *web server* 2 selama 1355 ms. Sehingga *web server* 2 memiliki waktu respon yang lebih cepat dari pada *web server* 1.
2. Pada analisa pengujian mekanisme *failover* sampel *request* yang dikirimkan sebanyak 200 *request* dan dilakukan sebanyak dua kali percobaan. Pada Tabel 4 dapat dilihat pada percobaan ke-1 menunjukkan bahwa ketika sedang terjadi perpindahan sistem dari *master server* ke *backup server request* tidak dapat masuk seluruhnya karena *master server* dalam keadaan mati. Pada percobaan ke-1 sistem memerlukan waktu respon rata-rata selama 1414 ms untuk berpindah dari *master server* ke *backup server* dan selama perpindahan tersebut terdapat 86 *request* yang berhasil dilayani dan 114 *request* yang gagal. Pada percobaan ke-2 perpindahan sistem terjadi dari *backup server* ke *master server*. Pada pengujian ini *request* yang berhasil masuk sebanyak 200 *request* dengan waktu respon rata-rata selama 1306.5 ms. Pengujian ini membuktikan bahwa sistem *load balancing* yang terpasang mampu melakukan *failover* untuk mengantisipasi kegagalan pada *server load balancer*.
3. Pada pengujian antar muka *website monitoring* sistem dapat menampilkan data dengan baik dan sesuai yang diharapkan. Aplikasi dapat menampilkan data jumlah r*equest* yang masuk dan pembagian beban *request* ke *web server* 1 dan *web server* 2. Aplikasi dapat menampilkan data dengan baik dalam bentuk grafik jumlah *request* dan waktu respon.

5. KESIMPULAN

Berdasarkan hasil perancangan, implementasi, dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. Penerapan metode *failover* terhadap *load balancer* pada penelitian ini telah berhasil. Modul bawaan dari Nginx yang pada dasarnya untuk melakukan *health check* dapat juga di gunakan untuk mendukung ketersediaan layanan *load balancer* dan mendukung mekanisme *failover*.
2. Sistem *load balancer* yang di terapkan juga berhasil membagi beban *request* ke *backend server* dengan berdasarkan pada algoritma *round robin*.
3. Rata-rata waktu respon di *web server* 1 selama 1358.2 ms, sedangkan di *web server* 2 1355 ms. Berdasarkan rata-rata tersebut dapat disimpulkan bahwa waktu respon pada *web server* 2 lebih cepat dari pada waktu respon pada *web server* 1.
4. Pada pengujian *failover* saat percobaan ke-1 *master server* dalam keadaan mati jumlah *request* yang berhasil masuk sebanyak 86 *request* dengan waktu respon selama 1414 ms, saat percobaan

ke-2 dengan kondisi *master server* aktif kembali jumlah *request* yang berhasil masuk sebanyak 200 *request* dengan waktu respon selama 1306,5 ms. Sehingga dapat disimpulkan bahwa waktu yang dibutuhkan dalam proses pemulihan kembali dari *backup server* ke *master server* lebih cepat dari pada waktu yang dibutuhkan saat koneksi *master server down* dan berpindah ke *backup server*.

6. SARAN

Berdasarkan kesimpulan yang diperoleh dalam penelitian ini, maka penulis memberikan beberapa saran yaitu:

1. Pada penelitian selanjutnya menggunakan jumlah *server* yang lebih banyak dengan spesifikasi yang sama agar komunikasi antar *server* dapat berjalan dengan baik.
2. Pada penelitian selanjutnya metode *failover* dapat diimplementasikan pada layanan yang lainnya selain HTTP misalnya MySQL.
3. Pada penelitian selanjutnya metode *failover* dapat diimplementasikan dengan menggunakan *tool* Keepalived yang menyediakan mekanisme khusus untuk menerapkan *failover*.

DAFTAR PUSTAKA

[1] B. J. Kuncoro, I. Ruslianto dan S. Bahri, "Implementasi Health Checks Monitoring Pada Load Balancing Di Sisi Web Server berbasis Android," *Jurnal Coding*, 2018.

[2] M. Rosalia, "Implementasi High Availability Server Menggunakan Metode Load Balancing dan Failover pada Virtual Web Server Cluster," *e-Proceeding of Enginering*, vol. 3, 2016.

[3] M. M. Luthfi, "Memahami Pentingnya Load Balancing," 26 December 2015. [Online]. Available: https://www.idcloudhost.com.

[4] G. Triono, "Implementasi Load Balancing Dengan Menggunakan Algoritma Round Robin Pada Kasus Pendaftaran Siswa Baru Sekolah Menengah Pertama Labschool Unesa Surabaya," *IDeaTech 2015*, pp. 169-176, 2015.

[5] M. M. Luthfi, "Mengenal Virtual Private Server atau VPS," idcloudhost, 29 Agustus 2015. [Online]. Available: http://www.idcloudhost.com.

[6] R. A. Setyawan, "Analisis Implementasi Load Balancing Dengan Metode Source Hash Scheduling Pada Protocol," *Jurnal EECCIS*, pp. 204-8, 2014.

[7] M. Data, "Instalasi Web Server Nginx pada Ubuntu 16.04," 2018. [Online]. Available: https://www.medium.com.

[8] GoAccess, "Home," 25 Januari 2021. [Online]. Available: https://www.goaccess.io.

[9] R. Dani dan F. Suryawan, "Perancangan dan Pengujian Load Balancing dan Failover menggunakan Nginx," *Khazanah Informatika*, vol. 3, pp. 43-50, 2017.

[10] Alimuddin dan A. Ashari, "Peningkatan Kinerja Siakad Menggunakan Metode Load Balancing dan Fault Tolerance Di Jaringan Kampus Universitas Halu Oleo," *IJCCS*, pp. 11-22, 2016.